

# Parametrized Algorithms for Random Serial Dictatorship

Haris Aziz<sup>a</sup>, Julián Mestre<sup>b</sup>

<sup>a</sup>*NICTA and UNSW, 223 Anzac Parade, Sydney, NSW 2033, Australia,  
Phone: +61 2 8306 0490*

<sup>b</sup>*School of Information Technologies, The University of Sydney, Australia,  
Phone: +61 2 9351 4276*

---

## Abstract

Voting and assignment are two of the most fundamental settings in social choice theory. For both settings, *random serial dictatorship (RSD)* is a well-known rule that satisfies anonymity, ex post efficiency, and strategyproofness. Recently, it was shown that computing the resulting probabilities is #P-complete both in the voting and assignment setting. In this paper, we present efficient parametrized algorithms to compute the *RSD* probabilities for parameters such as the number of agent types, alternatives, or objects. When the parameters are small, then the respective algorithms are considerably more efficient than the naive approach of going through all permutations of agents.

**Keywords:** Social choice theory, random serial dictatorship, random priority, computational complexity, assignment setting.

**JEL:** C6, C7.

---

## 1. Introduction

Voting and assignment are two of the most widely applied and important settings in social choice theory. In both settings, although one could also use *discrete* or *deterministic* rules, randomization is crucial to achieve minimal fairness requirements such as anonymity and neutrality. In voting, *agents* express preferences over *alternatives* and a *social decision scheme* returns a probability distribution over the alternatives based on the agents' preferences [5, 12, 17]. In the

---

*Email addresses:* haris.aziz@nicta.com.au (Haris Aziz),  
mestre@it.usyd.edu.au (Julián Mestre)

assignment setting, *agents* express preferences over *objects* and a *random assignment rule* returns a random assignment of the objects specifying the probability with which each object is allocated to each agent [6, 7, 9]. The objects are referred to as *houses* in the literature and the assignment setting is also known as *house allocation*. For the two settings, *RSD* is a desirable social decision scheme [4, 12] and random assignment rule [7, 10], respectively.

In the voting setting, *random serial dictatorship (RSD)* takes a permutation of agents uniformly at random and then selects an alternative by serially allowing agents in the permutation to refine the set of feasible alternatives. In the assignment setting, *RSD* takes a permutation uniformly at random and then lets the agents in the permutation serially take their most preferred house that has not yet been allocated.

For both the settings mentioned above, *RSD* is a well-known rule that is anonymous, strategyproof, and ex post efficient (randomizes over Pareto optimal alternatives). In fact, it has been conjectured to be the only rule that satisfies these properties [see e.g., 13, 16]. *RSD* is well-established and commonly used especially in resource allocation. In particular, the resulting probabilities of *RSD* can be viewed as fractional allocations in scheduling and other applications (in which the houses are in fact divisible) and hence important to compute [see e.g., 1, 7, 8, 10, 20]. Similarly, in voting, the probabilities returned by the *RSD* rule can be interpreted as fractions of time or resource allotted to the alternatives and hence crucial to compute. The probability of each alternative can also be used as a suggestion for the proportional representation of the alternative in representative democracy or seat allocation of a parliament [2, 21].

The definition of *RSD* for both the settings suggests natural exponential-time algorithms to compute the *RSD* probabilities: enumerate all the permutations and for each permutation, perform a linear number of operations. However these algorithms are naive and the question arises whether there are more efficient algorithms to compute the *RSD* probabilities.

Recently, Aziz et al. [3] showed that the resulting probabilities of *RSD* are #P-complete to compute both in the voting and the assignment settings. Independently, Saban and Sethuraman [18] also showed the same result for the assignment problem. In view of the inherent complexity of computing the *RSD* probabilities, Saban and Sethuraman [18] mentioned that identifying the conditions under which the problem is polynomial time as an open problem. Mennle and Seuken [14] propose random hybrid assignment mechanisms that hinge on the *RSD* probabilities. They termed the problem of computing *RSD* probabilities in the assignment domain as a “very difficult research problem.”

In view of the importance of *RSD* in both voting and resource allocation and the recent negative computational results, we undertake an algorithmic study of *RSD*. We show that *RSD* is amenable to efficient computation provided certain structural parameters are small. More precisely, we undertake a *parametrized* complexity analysis of *RSD* for both voting and assignment.

*Standard vs. parametrized complexity.* In *standard computational complexity theory*, only the size of the problem instance is considered as a measure of the problem’s complexity. An algorithm is deemed to be efficient if its running time is bounded by a polynomial function of the size of the instance; in other words, for every instance  $I$  of the problem, the running time of the algorithm is bounded by  $\text{poly}(|I|)$  where  $\text{poly}$  is a fixed polynomial independent of the instance and  $|I|$  is the size of the instance. Unfortunately, not every problem is known to admit an efficient algorithm. Indeed, researchers have identified certain classes of problems that are thought not to admit efficient algorithms. For example, the fact that *RSD* is #P-complete [3, 18] strongly suggests that there is no efficient algorithm for computing the *RSD* probabilities.

In *parametrized complexity theory*, a finer multivariate analysis is undertaken by considering multiple *parameters* of the problem instance [11, 15]. Intuitively, a parameter is some aspect of the problem input. For example, for computational problems on graphs, the maximum degree of the graph is a natural parameter. Let  $k$  be a parameter of an instance  $I$ . A problem with parameter  $k$  belongs to the class FPT, or is said to be *fixed-parameter tractable*, if there exists an algorithm that solves the problem in  $f(k) \cdot \text{poly}(|I|)$  time, where  $f$  is some computable function and  $\text{poly}$  is a polynomial both independent of  $I$ . The key idea behind FPT is to separate out the complexity into two components—a component  $\text{poly}(|I|)$  that depends solely on the size of the input, and a component  $f(k)$  that depends on the parameter. An FPT algorithm with parameter  $k$  can solve instances in which the input size of the instance is large as long as  $k$  is small and hence the growth of  $f(k)$  is relatively small. Of course if  $k$  is not small and  $f(k)$  is, say, exponential in  $k$ , then the running time of the FPT algorithm can become too slow for practical purposes. Nevertheless, designing FPT algorithms with different parameters is important as they expand tractability frontier for problems that are otherwise intractable in general.

*Contributions.* We propose algorithms to compute the *RSD* probabilities that under reasonable assumptions are significantly more efficient than the naive method of going over  $n!$  permutations of the agents. To be precise, we present FPT algorithms with parameters such as # agent types, # alternatives, # alternative types

and # houses. Many of our algorithms exploit different dynamic programming formulations where the recursion is based on new insights into *RSD* applied to voting and assignment. In this sense, our work not only yields more efficient algorithms for computing an *RSD* lottery, but also furthers our understanding of a keystone algorithm in social choice theory.

## 2. Voting Setting

We first define the voting setting and *RSD* formally. We follow the notation used in [4]. A voting setting consists of a set  $N = \{1, \dots, n\}$  of *agents* having preferences over a finite set  $A$  of *alternatives* where  $|A| = m$ . The preferences of agents over alternatives are represented by a preference profile  $R = (R_1, \dots, R_n)$  where each agent  $i \in N$  has complete and transitive preferences  $R_i$  over  $A$ . By  $(a, b) \in R_i$ , also denoted by  $a R_i b$ , we mean that alternative  $a$  is at least as preferred by agent  $i$  as alternative  $b$ . We denote with  $P_i$  the strict part of  $R_i$  ( $a P_i b$  if  $a R_i b$  but not  $b R_i a$ ), and with  $I_i$  the symmetric part of  $R_i$  ( $a I_i b$  if  $a R_i b$  and  $b R_i a$ ). A preference relation  $R_i$  is *linear* if  $a P_i b$  or  $b P_i a$  for all distinct alternatives  $a, b \in A$ . The size of an instance of a voting setting will be denoted by  $|R|$  which is equal to  $|N| \times |A|$ .

We let  $\Pi^N$  denote the set of all permutations of  $N$  and write a permutation  $\pi \in \Pi^N$  as  $\pi = \pi(1) \dots \pi(n)$ . If  $R_i$  is a preference relation and  $B \subseteq A$  a subset of alternatives, then  $\max_{R_i}(B) = \{a \in B : a R_i b \text{ for all } b \in B\}$  is the set of most preferred alternatives from  $B$  according to  $R_i$ . Let  $\pi(i)$  be the  $i$ -th agent in permutation  $\pi \in \Pi^N$ . Then,

$$RSD(N, A, R) = \sum_{\pi \in \Pi^N} \frac{1}{n!} \delta_{\text{uniform}}(\text{Prio}(N, A, R, \pi))$$

where

$$\text{Prio}(N, A, R, \pi) = \max_{R_{\pi(n)}} \left( \max_{R_{\pi(n-1)}} \left( \dots \left( \max_{R_{\pi(1)}}(A) \right) \dots \right) \right),$$

and  $\delta_{\text{uniform}}(B)$  is the uniform lottery over the multi-set  $B$ . In the literature,  $\text{Prio}(N, A, R, \pi)$  is simply referred to as *serial dictatorship* with respect to ordering  $\pi$ . We illustrate how *RSD* works with the aid of a simple example.

**Example 1** (Illustration of *RSD* in voting). *Consider the following preference profile.*

$$\begin{aligned}
1 : & \quad a \ I_1 \ b \ I_1 \ c \ P_1 \ d \\
2 : & \quad b \ I_2 \ d \ P_2 \ a \ P_2 \ c \\
3 : & \quad c \ P_3 \ a \ I_3 \ b \ I_3 \ d
\end{aligned}$$

Then let us consider the *Prio* outcomes for each permutation over the voters.

$$\begin{array}{ll}
123 : & \{b\} & 132 : & \{c\} \\
213 : & \{b\} & 231 : & \{b\} \\
312 : & \{c\} & 321 : & \{c\}
\end{array}$$

Thus the *RSD* lottery is  $[a : 0, b : 1/2, c : 1/2]$ .

If each agent has a unique most preferred alternative, the *RSD* lottery can be computed in linear time: the *RSD* probability of an alternative is the fraction of agents who express it as most preferred. However, the problem of computing *RSD* probabilities become #P-complete when agents do not express strict preferences. As seen by the formal definition as well as the example above, *RSD* probabilities can be computed by enumerating all the  $n!$  permutations over agents. We present alternative algorithms to compute the *RSD* probabilities that are significantly faster provided certain structural parameters are small.

Two agents are said to be of the same *type* if they have identical preferences. Two alternatives are of the same *type* if every agent is indifferent between them. The following lemma shows that without loss of generality we can focus on instance where no two alternatives have the same type.

**Lemma 1.** *There is a linear-time reduction from general instances of RSD to simplified instances of RSD where no two alternatives have the same type.*

*Proof.* Given an instance  $(N, A, R)$ , we construct an equivalent *simplified* instance  $(N, A', R')$  by contracting all alternatives of the same type into a ‘super’ alternative.

Given an *RSD* lottery for  $(N, A', R')$  we can construct a lottery for  $(N, A, R)$  by uniformly dividing the *RSD* probability of the ‘super’ alternative among the alternatives in  $A$  that induced it.  $\square$

Unless otherwise stated, from now on, we assume that we are dealing with *simplified* instances where no two alternatives have the same type.

Let  $a \in A$  be a fixed but arbitrary alternative. For each  $i \in N$ , we define the signature of  $i$  (with respect to  $a$ ) to be  $(C, D)$  where  $C$  is the subset of alternatives that are as good as  $a$ , and  $D$  is the subset of alternatives that are strictly better than  $a$ ; more formally,  $C = \{b : b I_i a\}$  and  $D = \{b : b P_i a\}$ . Notice that even if two agents have different types, they can still have the same signature. On the other hand, if two agents have the same type, they must have the same signature.

For notational convenience we enumerate all the signatures and denote the set of signatures with  $\mathcal{S} = \{1, 2, \dots\}$ . Let the  $i$ th signature be defined by the pair  $(C_i, D_i)$  and let  $t_i > 0$  be the number of agents having this signature. For a subset  $X \subseteq \mathcal{S}$  of signatures we use  $t(X)$  to denote  $\sum_{i \in X} t_i$ . Since we are dealing with a simplified instance, at least one agent is not indifferent between a given pair of alternatives and hence each Prio outcome results in a singleton set. Hence, it follows that  $\cap_{i \in N} C_i = \{a\}$ . In this way, running the serial dictatorship on any permutation of the agents either does not select  $a$  or selects exactly  $a$ .

We show an FPT algorithm for computing  $RSD(R)(a)$  where the parameter is the number of signatures  $|\mathcal{S}|$ .

**Theorem 1.** *In the voting setting, for each alternative  $a \in A$  there is an algorithm for computing  $RSD(R)(a)$  that runs in  $O(|R| + m|\mathcal{S}| \cdot 2^{|\mathcal{S}|})$  time.*

*Proof.* For each  $X \subset \mathcal{S}$  we define a residual problem where the set of alternatives is  $\cap_{i \in \mathcal{S} \setminus X} C_i$ , where there are no agents with a signature in  $\mathcal{S} \setminus X$ , and where we still have  $t_i$  agents with signature  $i$  for each  $i \in X$ . For  $X = \mathcal{S}$  the residual problem is the same as the original problem. In the residual problem defined by  $X$  we say a permutation of its agents is *lucky* if running the serial dictatorship with this permutation selects alternative  $a$ . Our algorithm is based on a dynamic programming formulation for counting the number of lucky permutations:

$$M[X] = \# \text{ lucky permutations in the residual problem defined by } X. \quad (1)$$

Notice that  $RSD(R)(a) = M[\mathcal{S}]/n!$ , so if we can compute  $M$ , we are done. In the rest of the proof, we derive a recurrence to do just that.

For each  $X \subset \mathcal{S}$  we define the set of *admissible signatures* to be

$$\phi(X) = \left\{ i \in X : D_i \cap \bigcap_{j \in \mathcal{S} \setminus X} C_j = \emptyset \right\}$$

and

$$\phi(\mathcal{S}) = \{i \in \mathcal{S} : D_i = \emptyset\}.$$

The key observation is that every lucky permutation in the residual problem defined by  $X$  must start with an agent having an admissible signature.

To compute the value of the DP (dynamic program) states, we use the following recurrence.

$$M[X] = \begin{cases} t(X)! & \text{if } \phi(X) = X \\ 0 & \text{if } \phi(X) = \emptyset \\ \sum_{i \in \phi(X)} t_i! \binom{t(X) - 1}{t_i - 1} M[X \setminus \{i\}] & \text{otherwise} \end{cases} \quad (2)$$

Let us briefly justify each case of the recurrence. First, consider the case  $\phi(X) = X$ , which means that in the residual problem defined by  $X$  every agent has  $a$  in its top equivalence class. If that is the case, then every permutation of the agents is lucky. Since there are  $t(X)$  agents in the residual problem, it follows that there are  $t(X)!$  lucky permutations.

Second, consider the case  $\phi(X) = \emptyset$ , which means that in the residual problem defined by  $X$ , not a single agent has  $a$  in its top equivalence class. If that is the case, then there are no lucky permutations.

Finally, consider the case  $\emptyset \subset \phi(X) \subset X$ . Recall that every lucky permutation must begin with an agent with a signature  $i \in \phi(X)$ . Notice that after such an agent is chosen the set of possible alternatives is reduced to  $\bigcap_{j \in \mathcal{S} \setminus (X \cup \{i\})} C_j$ . Also, the remaining agents ( $t_i - 1$ ) agents with signature  $i$  do not further constrain the set of alternatives. Therefore, if we take a lucky permutation in the residual problem defined by  $X$  and we strip from it all agents with signature  $i$ , we are left with a lucky permutation for the residual problem defined by  $X \cup \{i\}$ . Similarly, if we take a lucky permutation for the residual problem defined by  $X \cup \{i\}$  and we prepend one agent with signature  $i$  and insert the remaining  $t_i - 1$  agents with signature  $i$  any way we want, we have a lucky permutation for the residual problem defined by  $X$ . Notice that for each lucky permutation for  $X \cup \{i\}$  there are  $\binom{t(X) - 1}{t_i - 1}$  ways of choosing the positions for the agents with signature  $i$  and for each one of those there are  $t_i!$  ways of distributing the individual agents. It follows that there are  $\sum_{i \in \phi(X)} t_i! \binom{t(X) - 1}{t_i - 1} M[X \setminus \{i\}]$  lucky permutations for the residual problems defined by  $X$ .

Computing the signatures  $\mathcal{S}$  and their frequencies  $t_1, \dots, t_{|\mathcal{S}|}$  takes  $O(|R|)$  time, where  $|R|$  is the total length of the agent preferences. Computing the admissible signature function  $\phi$  can be done in  $O(m |\mathcal{S}| \cdot 2^{|\mathcal{S}|})$  time, where  $m$  is the number of alternatives. The size of the DP table is  $2^{|\mathcal{S}|}$ , and given all the previous

information, computing each entry of the table takes  $O(|\mathcal{S}|)$  time. Hence, the total time to compute  $RSD(R)(a)$  is  $O(|R| + m|\mathcal{S}| \cdot 2^{|\mathcal{S}|})$ .  $\square$

**Corollary 1.** *There is an FPT algorithm for computing  $RSD(R)(a)$  with parameter  $n = \#$  of agents. The running time is  $O(|R| + mn \cdot 2^n)$ .*

*Proof.* Two agents of the same type must have the same signature, so  $|\mathcal{S}| \leq n$ . It follows from Theorem 1 that the running time is  $O(|R| + mn \cdot 2^n) = O(|R| \cdot (1 + 2^n)) = O(\text{poly}(|R|) \cdot (1 + 2^n))$  and hence the algorithm is FPT with parameter  $n$ .  $\square$

**Corollary 2.** *There is an FPT algorithm for computing  $RSD(R)(a)$  with parameter  $T = \#$  of agent types. The running time is  $O(|R| + mT \cdot 2^T)$ .*

*Proof.* In the worst case, each agent type has a different signature, so  $|\mathcal{S}| \leq T$ . It follows from Theorem 1 that the running time is  $O(|R| + mT \cdot 2^T) = O(|R| \cdot (1 + T \cdot 2^T)) = O(\text{poly}(|R|) \cdot (1 + T \cdot 2^T))$  and hence the algorithm is FPT with parameter  $T$ .  $\square$

**Corollary 3.** *There is an FPT algorithm for computing  $RSD(R)(a)$  with parameter  $m = \#$  of alternatives. The running time is  $O(|R| + m \cdot 3^m \cdot 2^{3^m})$ .*

*Proof.* This follows from Theorem 1 and the fact that if there are  $m$  alternatives, there are at most  $3^m$  different signatures  $(C_i, D_i)$  because

$$\sum_{k=1}^m \binom{m}{k} 2^{m-k} = 3^m,$$

where the first term of the left-hand-side product is the number of way of choosing a subset  $C_i$  of size  $k$  and second term is how many choices we have for  $D_i$  given that  $|C_i| = k$  and  $C_i \cap D_i = \emptyset$ . Therefore,  $|\mathcal{S}| \leq 3^m$ .  $\square$

**Corollary 4.** *There is an FPT algorithm for computing  $RSD(R)(a)$  with parameter  $q = \#$  of alternative types. The running time is  $O(|R| + q \cdot 3^q \cdot 2^{3^q})$ .*

*Proof.* This follows from Lemma 1 and Corollary 3.  $\square$

### 3. Assignment Setting

An *assignment setting* is a triple  $(N, H, R)$ , where  $N$  is a set of  $n$  agents,  $H$  is a set of  $m \leq n$  houses, and  $R = (R_1, \dots, R_n)$  is a preference profile that contains,



for each agent  $i$ , a *linear* preference relation on some subset of houses that are *acceptable* to the agent. If a house is *unacceptable* to an agent, then he would prefer not to get any house than being allocated an unacceptable house. The size of an instance of an assignment setting will be denoted by  $|R|$  which is equal to  $|N| \times |H|$ . Every randomized assignment yields a *fractional assignment* that specifies, for every agent  $i$  and every house  $h$ , the probability  $p_{ih}$  that house  $h$  is assigned to agent  $i$ . The fractional assignment can be seen as a compact representation of the randomized assignment. *RSD* takes a permutation uniformly at random and then lets the agents in the permutation serially take their most preferred house that has not yet been allocated. If no acceptable houses remain, then the agent takes no house.

It is easily observed that the assignment setting is a special case of a social choice problem where  $A$ , the set of alternatives is the set of all discrete assignments and the preferences of agents over  $A$  are induced by their preferences over  $H$ . Although agents have strict preferences over the houses, they are indifferent among all assignments in which they are allocated the same house. We illustrate how *RSD* works as a random assignment rule.

**Example 2** (Illustration of *RSD* for random assignment). *Consider the following preference profile.*

$$\begin{aligned} 1 : & \quad a \ P_1 \ b \ P_1 \ c \\ 2 : & \quad a \ P_2 \ b \ P_2 \ c \\ 3 : & \quad b \ P_3 \ a \ P_3 \ c \end{aligned}$$

*Agent 1 and 2 are of the same type since they have the same preferences. Let us consider the Prio outcomes for each permutation over the voters. Each outcome is a discrete matching.*

$$\begin{array}{ll} 123 : & \{\{1, a\}, \{2, b\}, \{3, c\}\} & 132 : & \{\{1, a\}, \{3, b\}, \{2, c\}\} \\ 213 : & \{\{2, a\}, \{1, b\}, \{3, c\}\} & 231 : & \{\{2, a\}, \{3, b\}, \{1, c\}\} \\ 312 : & \{\{3, b\}, \{1, a\}, \{2, c\}\} & 321 : & \{\{3, b\}, \{2, a\}, \{1, c\}\} \end{array}$$

*Thus the *RSD* fractional assignment is obtained by taking the uniform convex combination of the discrete assignments for each of the permutations (see Table 1).*

Although the assignment setting is a subdomain of social choice (voting), it does not mean that positive algorithmic results for voting imply the same for the

	$a$	$b$	$c$
1	1/2	1/6	1/3
2	1/2	1/6	1/3
3	0	2/3	1/3

Table 1: Fractional/randomized assignment as a result of  $RSD$ .

assignment domain. The reason is that the transformation from an assignment setting to a corresponding voting problem leads to an exponential blowup in the number of alternatives. Hence, results in the previous section do not carry over directly to the domain of assignments.

Saban and Sethuraman [18] showed that it is not possible (under suitable complexity-theoretic assumptions) to design an efficient algorithm for approximating the  $RSD$  probabilities even if randomization is used. Since neither randomization nor approximation is helpful, this further motivates a parametrized algorithm approach. The main result in this section is an FPT algorithm with composite parameter number of houses and number of agent types. From this result we derive two corollaries: There is a polynomial-time algorithm when the number of agent types is constant, and an FPT algorithm with parameter number of houses.

We use  $T$  to denote the number of agent types, and  $d_j$  to denote the number of agents of type  $j$  in the instance. With a slight abuse of notation, we will use  $R_j$  to denote preference order of an agent of type  $j$ .

**Theorem 2.** *In the assignment setting, for each  $i \in N$  and  $h \in H$  there is an algorithm for computing  $RSD(R)(i)(h)$  running in  $O(|R| + \binom{m+T}{T} \cdot T \cdot m^{T+1})$  time.*

*Proof.* Let  $\vec{s} = (s_1, \dots, s_T)$  be a vector of integers where  $0 \leq s_j \leq d_j$  for all  $j = 1, \dots, T$ . Also let  $\vec{b} = (b_1, \dots, b_T)$  where  $b_j \in H \cup \{\text{nil}\}$  for  $j = 1, \dots, T$ . Multiple entries of  $\vec{b}$  can have the same house. Let  $\text{dom}(\vec{b})$  be those houses that are *dominated* by the allocation  $\vec{b}$ , namely, for each house  $h' \in \text{dom}(\vec{b})$ , there must be an agent type  $j$  that prefers  $h'$  to  $b_j$ ; more formally,

$$\text{dom}(\vec{b}) = \left\{ h' \in H : \exists j : \begin{array}{ll} b_j \neq \text{nil} & \wedge \quad h' P_j b_j, \text{ or} \\ b_j = \text{nil} & \wedge \quad h' \text{ is acceptable to type } j \end{array} \right\}.$$

For each  $(\vec{s}, \vec{b})$  we define a residual problem where the set of houses available

is  $H \setminus \text{dom}(\vec{b})$  and there are  $s_j$  agents of type  $j$  for each  $j = 1, \dots, T$ . Intuitively,  $\vec{b}$  is maintained in a way so that each  $b_j$  is the house most preferred by agent type  $j$  that has yet not been allocated. Let  $j^*$  be the type of agent  $i$  from the theorem statement. Consider a permutation of the agents where there are  $s_j$  agents of type  $j$  and  $i$  is one of the agents of type  $s_{j^*}$ —the precise identity of the other agents is not important. We say such a permutation is *lucky* if running serial dictatorship with this permutation on the residual instance results in assigning  $i$  to  $h$ .

We again use a dynamic programming formulation based on counting lucky permutations in the residual problems

$$M[\vec{s}, \vec{b}] = \# \text{ lucky permutations in the residual problem defined by } (\vec{s}, \vec{b}).$$

Our goal now is to derive a recurrence relation for  $M[\vec{s}, \vec{b}]$ . We begin with some base cases. First, if there are no agents of type  $j^*$  left or if the house  $h$  is not available anymore, then there are no lucky permutations; more formally,

$$M[\vec{s}, \vec{b}] = 0 \quad \text{if } s_{j^*} = 0 \text{ or } h \in \text{dom}(\vec{b}).$$

We say that  $\vec{b}$  is *degenerate* if  $b_j \in \text{dom}(\vec{b})$  for some  $j$ . For such degenerate vectors, let us define  $\text{closure}(\vec{b})$  to be the vector  $\vec{b}$  such that  $\tilde{b}_j$  is the most preferred house by agents of type  $j$  that does not belong to  $\text{dom}(\vec{b})$  or nil if all acceptable houses for agents of type  $j$  belong to  $\text{dom}(\vec{b})$ . It follows that if  $\vec{b}$  is degenerate then

$$M[\vec{s}, \vec{b}] = M[\vec{s}, \text{closure}(\vec{b})].$$

Let  $\text{next}_j(h')$  be the house that comes after  $h'$  in the total order of type  $j$  preferences. If  $h'$  happens to be the last acceptable house for type- $j$  agents then  $\text{next}_j(h') = \text{nil}$ . It is worth noting that  $\text{closure}(\vec{b})$  can be defined recursively in terms of the next operator:

$$\text{closure}(\vec{b}) = \begin{cases} \text{closure}(\vec{b}_{-j}, \text{next}_j(b_j)) & \text{if } \exists j : b_j \in \text{dom}(\vec{b}), \\ \vec{b} & \text{if } \forall j : b_j \notin \text{dom}(\vec{b}), \end{cases}$$

where the notation  $(\vec{b}_{-j}, x)$  denotes the vector that is identical to  $\vec{b}$  except for coordinate  $j$ , which takes the value  $x$ ; in other words,

$$(\vec{b}_{-j}, x) = (b_1, \dots, b_{j-1}, x, b_{j+1}, \dots, b_T).$$

Our final corner case is that where there is an agent of type  $j$  such that  $b_j = \text{nil}$  and  $s_j > 0$ . In this case it does not matter where the remaining  $s_j$  agents of type  $j$  are placed in the permutation since none of them will get a house. Therefore, we get

$$M[\vec{s}, \vec{b}] = \binom{s_1 + \dots + s_T}{s_j} s_j! \cdot M[(\vec{s}_{-j}, 0), \vec{b}].$$

For the recursive case of the recurrence, we condition on the type of agent that is chosen to lead the lucky permutation. If the agent is of type  $j \neq j^*$  and  $b_j \neq \text{nil}$ , assuming  $s_j > 0$ , there are  $s_j$  agents to choose from, so the number of such permutations will be

$$s_j \cdot M[(\vec{s}_{-j}, s_j - 1), (\vec{b}_{-j}, \text{next}_j(b_j))].$$

If the agent is of type  $j^*$ , assuming  $s_{j^*} > 0$ , and the agent is not  $i$ , then the number of such permutations is

$$(s_{j^*} - 1) \cdot M[(\vec{s}_{-j^*}, s_{j^*} - 1), (\vec{b}_{-j^*}, \text{next}_{j^*}(b_{j^*}))].$$

Finally, if the agent of type  $j^*$  leading the lucky permutation is  $i$  itself, then the number of such permutations is

$$\begin{cases} (-1 + \sum_j s_j)! & \text{if } b_{j^*} = h \\ 0 & \text{if } b_{j^*} \neq h. \end{cases}$$

Putting everything together we get the following recurrence for  $(\vec{s}, \vec{b})$  for the case when  $\vec{b}$  is non-degenerate,  $h \notin \text{dom}(\vec{b})$ ,  $s_{j^*} > 0$ :

$$\begin{aligned} M[\vec{s}, \vec{b}] = & \sum_{\substack{j \neq j^*: \\ s_j > 0 \wedge b_j \neq \text{nil}}} s_j \cdot M[(\vec{s}_{-j}, s_j - 1), (\vec{b}_{-j}, \text{next}_j(b_j))] \\ & + (s_{j^*} - 1) \cdot M[(\vec{s}_{-j^*}, s_{j^*} - 1), (\vec{b}_{-j^*}, \text{next}_{j^*}(b_{j^*}))] \\ & + \begin{cases} (-1 + \sum_j s_j)! & \text{if } b_{j^*} = h \\ 0 & \text{if } b_{j^*} \neq h \end{cases} \end{aligned} \quad (3)$$

Once the table is filled, the probability we are after is simply

$$RSD(R)(i)(h) = \frac{M[(d_1, \dots, d_T), (h_1, \dots, h_T)]}{n!},$$

where  $h_j$  is the house most preferred by agents of type  $j$ .

Let us bound the running time of the algorithm. First, we show how to efficiently compute the set of agent types present in the instance, their frequencies and preferences. Think of the linear preference relation of some agent as a “string” whose “letters” are houses. Notice that two agents of the same type give rise to the same string. In  $O(|R|)$  time we can generate the set of all string and build a trie [Ch. 5, 19] out of them, keeping a frequency count of how many strings of each kind we have seen, which correspond to how many agents of that type there are.

Next, we need to bound the time it takes to fill the DP table. Notice that the total number of entries  $M[\vec{s}, \vec{b}]$  can be as large as  $\prod_j d_j \cdot m^T$  entries, since there are  $\prod_j d_j$  choices for  $\vec{s}$  and  $m^T$  choices for  $\vec{b}$ . Unfortunately, this would be too large for our purposes. The key observation is that not all possible vectors  $\vec{s}$  are reachable from our recurrence. In particular, notice that every time we apply (3) we decrease  $\sum_j s_j$  by one and once all houses are assigned, the vector  $\vec{b}$  must be nil everywhere, which means we are at the base case of the recurrence. Therefore, we only need to keep track of vectors  $\vec{s}$  where  $\sum_j (d_j - s_j) \leq m$ . There are only  $\binom{m+T-1}{T-1}$  such vectors. Thus, the total number of entries we need to keep track of is bounded by  $\binom{m+T}{T} \cdot m^T$ . Computing  $M[\vec{s}, \vec{b}]$  using (3) takes  $O(T \cdot m)$  time, while using other cases of the recurrence takes  $O(m)$  time provided the function  $\text{closure}(\cdot)$  is already computed, which takes  $O(T \cdot m^{T+1})$  time overall. Adding everything up, we get that the total running time is  $O(|R| + \binom{m+T}{m} \cdot T \cdot m^{T+1})$  as it appears in the theorem statement.  $\square$

**Corollary 5.** *There is an FPT algorithm for computing  $RSD(R)(i)(h)$  with parameter  $m = \#$  of houses. The running time is  $O(|R| + m^{m^m})$ .*

*Proof.* Notice that when we have  $m$  houses, there can be at most  $\sum_{k=1}^m k!$  agent types, one for each total ordering of every subset of the  $m$  houses. The time bound follows from plugging  $T = m!(1 + \frac{2}{m}) \geq \sum_{k=1}^m k!$  into Theorem 2:

$$\begin{aligned}
\binom{m+T}{T} \cdot T \cdot m^{T+1} &= \binom{m+T}{m} \cdot T \cdot m^{T+1}, \\
&\leq \frac{(m+T)^m}{m!} \cdot m! \left(1 + \frac{2}{m}\right) \cdot m^{m!(1+\frac{2}{m})+1}, \\
&\leq \left(m + m! \left(1 + \frac{2}{m}\right)\right)^m \cdot \left(1 + \frac{2}{m}\right) \cdot m^{m!(1+\frac{2}{m})+1}, \\
&\leq \left(m! \left(1 + \frac{3}{m}\right)\right)^m \cdot \left(1 + \frac{2}{m}\right) \cdot m^{m!(1+\frac{2}{m})+1}, \\
&\leq m!^m \cdot \left(1 + \frac{3}{m}\right)^{m+1} \cdot m^{m!(1+\frac{2}{m})+1}, \\
&\leq m^{m^2} \cdot O(1) \cdot m^{m!(1+\frac{2}{m})+1}, \\
&= O(m^{m^m}),
\end{aligned}$$

where the last inequality follows from the upper bound  $m! \leq e(m^{0.5}) \left(\frac{m}{e}\right)^m$ , which is an easy consequence of Stirling's approximation for factorial numbers.  $\square$

**Corollary 6.** *There is an algorithm for computing  $RSD(R)(i)(h)$  whose running time is  $O(|R| + T \cdot m^{2T+1})$ .*

*Proof.* The time bound follows directly from Theorem 2:

$$\begin{aligned}
\binom{m+T}{T} \cdot T \cdot m^{T+1} &\leq \frac{(m+T)^T}{T!} \cdot T \cdot m^{T+1} \\
&\leq (m+1)^T \cdot T \cdot m^{T+1} \\
&= O(T \cdot m^{2T+1}).
\end{aligned}$$

$\square$

#### 4. Conclusions

In this paper, we presented the first parametrized complexity analysis of  $RSD$  both for the voting and assignment setting. For voting, we presented FPT algorithms for parameters # agent types and # alternatives. For the assignment setting,

Setting	Parameter	Complexity	Reference
Voting	$n$ : # agents	in FPT $O( R  + mn \cdot 2^n)$	Cor. 1
Voting	$T$ : # agent types	in FPT $O( R  + mT \cdot 2^T)$	Cor. 2
Voting	$m$ : # alternatives	in FPT $O( R  + m \cdot 3^m \cdot 2^{3^m})$	Cor. 3
Voting	$q$ : # alternative types	in FPT $O( R  + q \cdot 3^q \cdot 2^{3^q})$	Cor. 4
Assignment	$m$ : # houses	in FPT $O( R  + m^{m^m})$	Cor. 5
Assignment	$T$ : # agent types	$O( R  + T \cdot m^{2T+1})$	Cor. 6

Table 2: Parametrized time complexity bounds for  $RSD$

we presented an FPT algorithm for parameter # houses. Although an FPT algorithm for the assignment setting with parameter # agent types still eludes us, we showed that the problem is polynomial-time solvable if # agent types is constant. We leave as an open problem to settle the parametrized complexity of  $RSD$  in the assignment setting for the parameter  $p = \#$  of agent types.

## Acknowledgments

The authors thank the anonymous reviewers whose comments helped to improve the presentation of the paper. NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

## References

- [1] A. Abdulkadiroğlu and T. Sönmez. Random serial dictatorship and the core from random endowments in house allocation problems. *Econometrica*, 66(3):689–702, 1998.
- [2] H. Aziz. Maximal Recursive Rule: A New Social Decision Scheme. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 34–40, 2013.
- [3] H. Aziz, F. Brandt, and M. Brill. The computational complexity of random serial dictatorship. *Economics Letters*, 121(3):341–345, 2013.
- [4] H. Aziz, F. Brandt, and M. Brill. On the tradeoff between economic efficiency and strategyproofness in randomized social choice. In *Proceedings of*

- the 12th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 455–462. IFAAMAS, 2013.
- [5] S. Barberà. Majority and positional voting in a probabilistic framework. *Review of Economic Studies*, 46(2):379–389, 1979.
  - [6] A. Bhargat, D. Chakrabarty, and S. Khanna. Social welfare in one-sided matching markets without money. In *Proceedings of APPROX-RANDOM*, pages 87–98, 2011.
  - [7] A. Bogomolnaia and H. Moulin. A new solution to the random assignment problem. *Journal of Economic Theory*, 100(2):295–328, 2001.
  - [8] E. Budish and E. Cantillion. The multi-unit assignment problem: Theory and evidence from course allocation at Harvard. *American Economic Review*, 102(5):2237–2271, 2012.
  - [9] E. Budish, Y.-K. Che, F. Kojima, and P. Milgrom. Designing random allocation mechanisms: Theory and applications. *American Economic Review*, 103(2):585–623, 2013.
  - [10] H. Crès and H. Moulin. Scheduling with opting out: Improving upon random priority. *Operations Research*, 49(4):565–577, 2001.
  - [11] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
  - [12] A. Gibbard. Manipulation of schemes that mix voting with chance. *Econometrica*, 45(3):665–681, 1977.
  - [13] T. Lee and J. Sethuraman. Equivalence results in the allocation of indivisible objects: a unified view. August 2011.
  - [14] T. Mennle and S. Seuken. Partially strategyproof mechanisms for the assignment problem. Technical Report arXiv:1303.2558, arXiv.org, 2013.
  - [15] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
  - [16] D. C. Parkes and S. Seuken. *Economics and Computation*. 2013.



- [17] A. Procaccia. Can approximation circumvent Gibbard-Satterthwaite? In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*, pages 836–841. AAAI Press, 2010.
- [18] D. Saban and J. Sethuraman. The complexity of computing the random priority allocation matrix. In Y. Chen and N. Immorlica, editors, *Proceedings of the 9th International Workshop on Internet and Network Economics (WINE)*, Lecture Notes in Computer Science (LNCS), <http://www.columbia.edu/~js1353/pubs/rpcomplexity.pdf>, 2013.
- [19] R. Sedgewick and K. Wayne. *Algorithms*. Addison-Wesley, 4th edition, 2011.
- [20] L.-G. Svensson. Queue allocation of indivisible goods. *Social Choice and Welfare*, 11:323–330, 1994.
- [21] G. Tullock. *Towards a Mathematics of Politics*. University of Michigan Press, 1967.